Sponsored by:

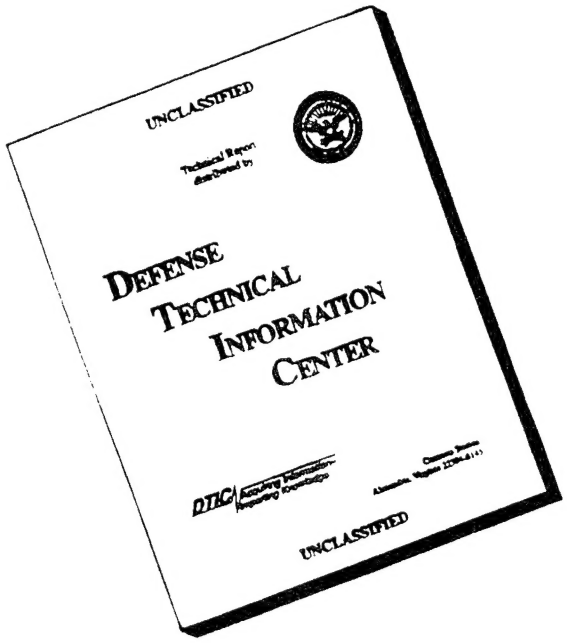
GIS/LIS '95

Annual Conference & Exposition

November 14-16, 1995
Nashville Convention Center
Nashville, Tennessee

Proceedings
Volume 1

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

IMPLEMENTATION OF A RULE-BASED FRAMEWORK FOR MANAGING UPDATES IN AN OBJECT-ORIENTED VPF DATABASE

David K. Arctur, GeoPlan Center
431 Arch Building, University of Florida
Gainesville, FL 32611-2004
email: arctur@ufl.edu

Co-authors:
Eman Anwar, University of Florida
Sharma Chakravarthy, University of Florida
Maria Cobb, Naval Research Laboratory
Miyi Chung, Naval Research Laboratory
Kevin Shaw, Naval Research Laboratory
John F. Alexander, University of Florida

ABSTRACT

This paper presents an approach for supporting reactive capability in an object-oriented GIS database, through the use of an event interface comprising an event generator and rule objects. This interface supports specification of events spanning sets of geographic-feature objects, and detection of primitive and complex events. Rules can be specified to apply either at a class level (i.e., to all instances of a given geographic-feature class) or at an instance level. In addition we allow evaluation of both pre- and post-conditions on changes to a feature. This approach is relevant in three distinct situations: (1) *immediate mode*, to execute rules immediately before or after some state change; (2) *deferred mode*, to execute rules at the end of several changes; and (3) *detached mode*, to perform rule-based actions separately from the state changes. GIS presents a rich set of problems for which this approach can be useful. This paper outlines the key elements of the rule-based approach employed in an object-oriented framework used for viewing and editing Vector Product Format (VPF) source data.

INTRODUCTION

A joint project of the U.S. Naval Research Laboratory (NRL) and the University of Florida during the past two years has resulted in the development of an object-oriented prototype application for viewing and managing Vector Product Format (VPF) geographic databases (DMA 1993), using the Smalltalk language on a Unix platform. As defined by the Defense Mapping Agency (DMA), the VPF specification uses a relational database approach for storing metadata, feature attributes and geo-spatial information. However, the current project uses an object-oriented representation of VPF (OVPF) to facilitate complex, user-interactive operations with the database (Arctur 1995a,b, Shaw 1994).

Building on the literature of recent developments in rule-based frameworks for object-oriented databases, we have implemented such a framework in our Smalltalk OVPF viewer/editor application to help in managing application-dependent constraints on changes to geographic features (e.g., preventing the user from creating or moving a land feature over water). We first present background on the rule-based framework, then introduce our implementation and provide an example of usage.

CONCEPTUAL BACKGROUND

Active Objects and Databases

During the past years database management systems (DBMS) have undergone dramatic changes as a result of the increasing requirements of modern day applications. Conventional record-oriented database systems are subject to the limitations of a finite set of data types and the need to normalize data. These limitations have led to the evolution of a new paradigm, namely object-oriented database management systems (ODBMS), which offer increased modeling power, flexible abstract data-typing facilities and the ability to encapsulate data and operations via the message* metaphor. Despite the ability to model complex objects and relationships, these ODBMSs still lack some of the requirements of a large class of new applications, specifically those requiring monitoring of situations and the ability to respond automatically, possibly subject to timing constraints.

Active databases have been proposed to meet some of the requirements of non-traditional applications (Chakravarthy 1994a). Active ODBMSs extend the normal functionality of ODBMSs with support for monitoring user-defined situations and reacting to them without user or application intervention. These ODBMSs continuously monitor situations to initiate appropriate actions in response to database updates, occurrence of particular states, or transition between states, possibly within a response-time window. The emergence of this trend of active ODBMSs serves a large variety of applications such as GIS, AM/FM, computer integrated manufacturing (Honnavalli 1994), process control, battle management, and network management. Furthermore, active databases provide an elegant means for supporting integrity constraints, access control, maintenance of derived data, and materialized views and snapshots.

Active behavior is by no means a new notion. However, it has been used to connote different behavior in various contexts within computer science. Morgan used the term "active database," perhaps for the first time (Morgan 1983), to describe a system that supports automatic update of views and derived data as base data are updated. In the artificial-intelligence community the term "active object" is used either for active knowledge representation and inference mechanisms or for achieving intelligent behavior and concurrent computation. The programming-language community uses the term "active object" in order to structure concurrent applications in an object-oriented programming language. Ishikawa used the term "active object" to distinguish real-time objects from others which have timing constraints (Ishikawa 1990). In summary, the term "active" has been used to convey concurrency, asynchronous behavior, and parallelism of active objects, intelligent behavior of agents/actors, or active capability of a system. In other literature similar notions are elaborated without using the term active explicitly.

The key distinction we draw between an active and a passive object lies in an active object's ability to monitor its state and take pre-defined actions that are based on the state changes. This is in constrast to a conventional object which responds to a message with a

* All actions performed in an object-oriented system are the result of sending a message to an object. The receiver-object then responds by executing a method by that name. For improved readability in this paper, we use the terms *message* and *method* interchangeably. However, these terms have distinct meanings; i.e., for a given *message* there may be one or more *methods* defined, as any number of objects can have a method with the same name.

predefined effect (of course based on the state), but the object cannot monitor its or other objects' status. This concept (of activeness), to some extent, is present in the actor model of Agha, et al. (Agha 1986).

Rules

Rules, also referred to as triggers and alerters (Chakravarthy 1989, Dayal 1988, Dittrich 1986, Su 1988), have been proposed to provide active functionality in ODBMSs. Rules, in the context of an active DBMS, consist primarily of three components: an event, a condition, and an action. An event is an indicator of a happening (either simple or complex). Events are recognized by the system or signalled by the user. For example, events such as the creation of an instance, the change of an attribute's value, and accessing an attribute's value are detected by the ODBMS. The condition specifies an optional predicate over the database state which is evaluated when its corresponding event occurs. The conditions to be monitored may be arbitrarily complex and may be defined not only on single data values or individual database states, but also on sets of data objects, transitions between states of materialized/derived objects, trends and historical data. Actions are the operations to be performed when an event occurs and its associated condition evaluates to true. Actions can be programs whose execution may in turn cause other events to occur. Once rules are specified declaratively to the system, it is the system's responsibility to monitor the situations (event-condition pairs) and execute the corresponding action when the condition is satisfied without any user or application intervention. The advantage of using rules as a means of providing active behavior is the freedom from explicitly hard-wiring code which checks the situations being monitored in each program that updates the database.

Events

Incorporation of rules in any system entails identifying what constitutes an event, developing an expressive event specification language, constructing an event detection mechanism, and identifying how to represent conditions and actions. Our framework is based on the classifications in Snoop (Chakravarthy 1994b,c) which defines semantics for various events and event operators in an object-oriented environment. An event is defined as something that happens at a point in time. In an object-oriented context, the events of interest are concerned with changes to an object's state. An object's state changes as the result of an update operation. Update operations occur when an object receives a message. Therefore, we view *each message sent* to an object as a potential event. Considering messages sent to objects as events per se is ambiguous; it is not clear whether the event is raised before or after the execution of the update message. To resolve this ambiguity, the *pre* and *post* clauses are introduced. The "pre" clause indicates the signalling of an event before the message is executed, while the "post" clause indicates the signalling of an event after the execution of the message.

Events are categorized as being either primitive or complex. *Primitive events* are those signalled at the beginning or at the end of execution of a single specific message. The term *beginning* refers to the point before the receipt of the message and *end* refers to the point after executing all operations within the method including the return statement. It is important to note that messages sent to objects are considered as primitive events regardless of the type of operations performed by the method.

Many applications are not well served by primitive events alone. *Complex events* (also called composite events) provide a simple and powerful mechanism for expressing the conjunction, disjunction and sequence of either primitive or other complex events. In our

framework an event can be defined as the *conjunction* of two events E1 and E2 which is signalled when both E1 and E2 occur, regardless of the order of execution. The *disjunction* of two events would be signalled when either E1 or E2 occurs. The *sequence* event would be signalled by completion of the sequential occurrence of a set of events. Further details on the event specification language, as well as descriptions of temporally-complex events may be found in (Chakravarthy 1994c).

RULE-BASED FRAMEWORK IMPLEMENTATION

The foregoing discussion provides a basis for introducing our implementation of the rule-based framework. As events are the most central part of the design, we will first describe event objects, then describe rule objects, and present an example of usage. We will then describe the outer framework in which event detection and rule firing take place.

Event Objects

Events are first-class objects in this framework as they have significant state and behavior. The PrimitiveEvent class in Figure 1 defines an eventMsg attribute which is inherited by all its subclasses. For each new instance of any event, this attribute is assigned the name of the message for which the event is raised. The ComplexEvent class defines further attributes used by its own subclasses.

The key method for each of the event classes is notify:. This method takes only one argument which specifies the name of the message which causes an event to be raised. The event is raised when the object(s) associated with the event object receives that message. For PrimitiveEvents the notify: method simply compares the argument to its own eventMsg attribute value and returns true if they match. For each ComplexEvent subclass, the notify: method also examines a particular combination of the status of its other attributes, before returning true or false.

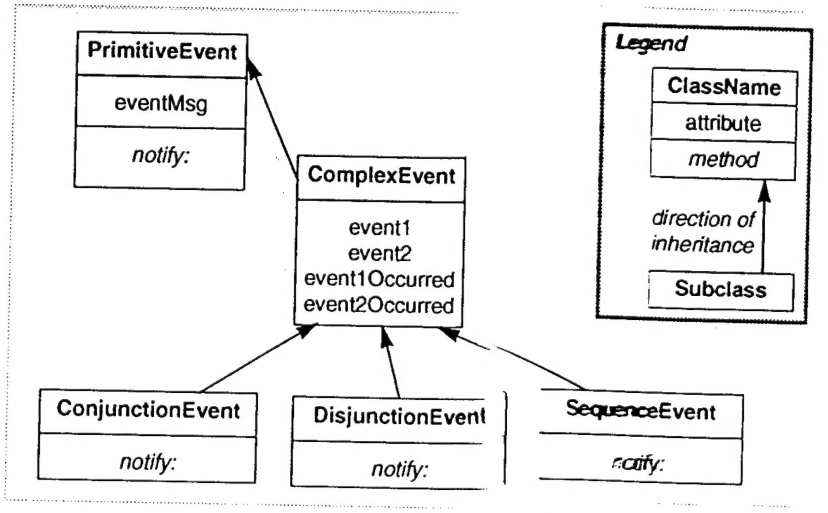


Figure 1. Event Class Hierarchy

An event instance is typically created at the time of rule creation. We will describe the Rule object and then present an example of usage.

Rule Objects

Rule objects have the structure shown in Figure 2. A single class suffices for defining all rules. The feature attribute may be assigned a pointer to either a single geographic-feature instance, such as a road or lake; or to a feature class, such as the defining class for

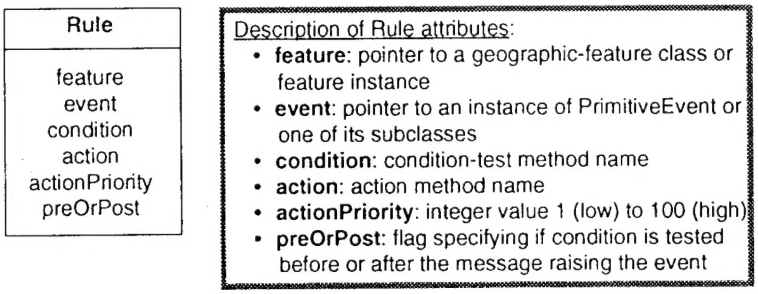


Figure 2. Structure of a Rule Object

roads or lakes. In the former, the rule will be applied only to a particular instance whereas in the latter, the rule will be applied to all instances of the defining class. The event attribute is assigned a pointer to a specific event instance (introduced above), which could be either a PrimitiveEvent or a ComplexEvent. The condition attribute is assigned the name of a method to be executed at the time the event is signalled, which will return true if the condition is met and false otherwise. The action method is then executed if the condition evaluates to true. The preOrPost attribute specifies the relative timing for execution of the condition method with respect to the message raising the event. The condition may be evaluated either before the event message is executed, or upon completion and return from the event message execution. The actionPriority attribute value is used to help mediate in situations where multiple rules fire at the same time.

Example Rule and Event Objects

An example of a Rule to prevent any BuildingPoint geographic features from being created over water is shown in Figure 3. In this case, the Rule is associated with the BuildingPoint class and thus will be applied to all instances of that class. Alternatively, the

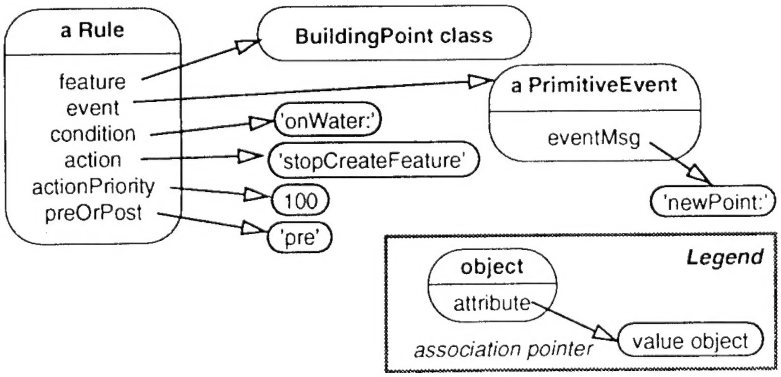


Figure 3. Example Rule and Event Objects

user may associate the Rule with a particular BuildingPoint instance. Due to the setting of the preOrPost attribute, the condition method onWater: is evaluated *before* the Event's eventMsg (the newPoint: method) is carried out. If the condition method onWater: returns true, the action method stopCreateFeature will then be executed, which will prevent the eventMsg method newPoint: from being performed. The actionPriority setting ensures this action will have highest priority among any other Rules which may also fire.

FeatureConstructor Objects

At this point we need to introduce the rest of the framework in which Events are detected and Rules are fired. In the OVPF viewer/editor tool, all changes to geographic-feature objects are handled through the use of FeatureConstructor objects (see Figure 4a). OVPF uses a construction-script framework with a state machine, supporting asynchronous events for flexibility in working with runtime-dependent constraints on changes to a given feature. This framework also is capable of extending its own semantics at runtime. It is beyond the scope of this paper to fully describe the FeatureConstructor framework, thus a very simplified portion of it is shown here for discussion.

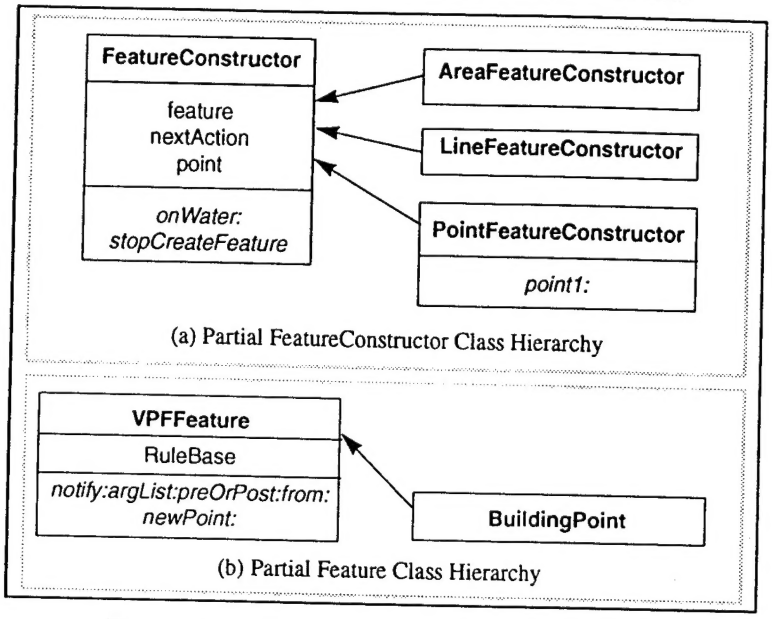


Figure 4. Key Components of Event Detection Framework

With reference to our example for creating a new BuildingPoint feature, we assume a Rule-Event pair has already been created (for checking if a new point feature is over water) and stored in VPFFeature's RuleBase (Figure 4b). This rule base is actually a persistent collection held in the ODBMS; its reference in VPFFeature is provided for convenient access at runtime. The following sequence of events could then take place at the user's initiation (step numbers correspond to those in Figure 5):

1. The user chooses the appropriate OVPF menu option to add a new geographic feature, and selects BuildingPoint from a list of available feature classes.

2. The OVPF graphical user interface (GUI) creates a PointFeatureConstructor.

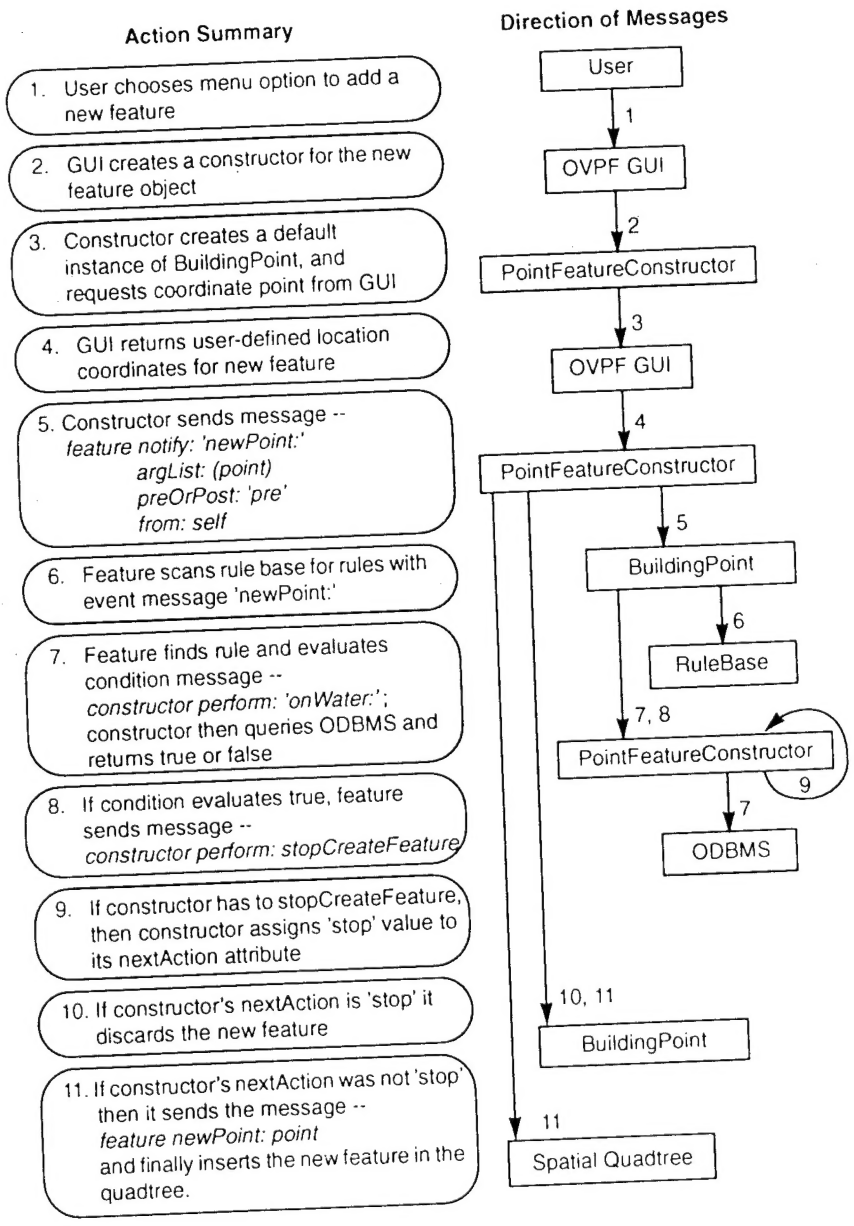


Figure 5. Flow of Control and Behavior For Rule-Event Example

3. The Constructor creates a default BuildingPoint feature object, and initiates a request to the GUI for a user-selected location coordinate point, to be returned via the point1: message.

4. On instruction from the GUI, the user chooses a location on the map with the mouse, and the GUI returns it as the argument in the point1: message to the Constructor.

5. Within its point1: method, the Constructor notifies the new BuildingPoint feature instance of an impending Event via the parameterized notify:argList: preOrPost:from: message.

6. The new BuildingPoint object executes the inherited notify:argList:preOrPost:from: method, which checks the rule base for all Rule-Event pairs whose eventMsg matches the notify: argument, in this case newPoint:.

7. If a matching Rule-Event pair is found, then the Rule's condition value (onWater:) is sent as a message to the Constructor to perform. The Constructor's onWater: method checks the database for any water-related features within a given tolerance of the user-selected coordinates, and returns true or false. By user's preference, this check can be performed either on just the features currently being displayed, or on features from all coverages in the ODBMS.

8. If the onWater: method returns true (coincident water feature was found), the Rule's action message is then sent to the Constructor. In this case if water features were found, the message stopCreateFeature would be the action message sent to the Constructor. Note that in the present framework, all applicable conditions are evaluated before any actions are performed. If multiple conditions return true, their action messages are sent to the PointFeatureConstructor in order of decreasing actionPriority.

9. If the Constructor receives the message stopCreateFeature, it will set its nextAction attribute to 'stop'.

10. Upon completion of all applicable conditions and actions, the new BuildingPoint object returns from executing the notify:argList:preOrPost:from: method. The thread of control reverts to the Constructor's point1: method, which then checks its nextAction setting. If it is 'stop' then the new default BuildingPoint feature is discarded, and control returns to the user with a descriptive dialog message.

11. If the nextAction is not 'stop' then the Constructor sends the newPoint: message to the new BuildingPoint, inserts it in the spatial quadtree, and presents the user with a dialog window to fill in any BuildingPoint feature attributes needed.

SUMMARY AND PROSPECTS

The GIS field presents a rich set of problems for which this approach can be useful. It can be used in three distinct situations: (1) *immediate mode*, to execute rules immediately before or after some state change; (2) *deferred mode*, to execute rules at the end of several changes; and (3) *detached mode*, to perform rule-based actions separately from the state changes. Furthermore, it has the advantage over traditional inference-engine approaches

in that it will work with an arbitrarily-large database of persistent objects, rather than being limited to those objects which can fit in memory.

The rule-event framework and procedures were surprisingly simple to implement. The FeatureConstructor classes, together with a single supporting method in VPFFeature class (notify:argList:preOrPost:from:), provide a simple and flexible event detection and rule processing system. While it introduces some processing overhead, all but the spatial query to the ODBMS in step 7 are very fast operations. An important benefit of this object-oriented framework is the potential for direct reuse by other FeatureConstructors of condition checks and actions such as the onWater: and stopCreateFeature methods. Furthermore, with this system provision can be made for adding and changing rules at runtime.

The design presented here is easily extended to trigger on any kind of change (create, modify, delete) to geographic-feature objects, as well as to specific feature attributes and spatial coordinates of a given feature object. This could be a significant advantage over the triggers supported by many commercial relational and even hybrid object-relational DBMSs. These DBMSs can typically trigger only on insert, update or delete of a complete feature record, rather than being able to discriminate on changes made to a single feature attribute.

There are a number of issues related to supporting rules that have not been addressed in this paper. We mention these for completeness as well as to provide directions for further areas of research. The number of rules defined in an application can become very large and may be defined by various users at different points in time. This can lead to the problem of having inconsistent or conflicting rules present within the application. For example, user A may define a rule R1 whose action may trigger rule R2 defined by user B. Suppose rule R2's action results in triggering rule R1, thereby yielding an infinite loop. From this scenario, it is evident that a mechanism for establishing the consistency or correctness of rules must be an inherent part of any active system. This involves writing algorithms which statically detect rule conflicts as well as algorithms which dynamically detect problems such as infinite rule triggering. We are in the early stages of pursuing these next steps.

ACKNOWLEDGMENT

We wish to thank the U.S. Defense Mapping Agency for supporting this work.

REFERENCES

Agha, G. 1986. "Actors: A Model of Concurrent Computation in Distributed Systems", MIT Press, Cambridge MA.

Anwar, E., Maugis, L. and Chakravarthy, S. 1993. "A New Perspective on Rule Support for Object-Oriented Databases" in Proceedings of ACM SIGMOD, May 1993, pp. 99-108.

Arctur, D., et al. 1995a. "OVPF Report: Object-Oriented Database Design Issues," Interim Project Report to DMA, Naval Research Laboratory, June 1995.

Arctur, D., et al. 1995b, "Comparison and Benchmarks for Import of Vector Product Format (VPF) Geographic Data from Object-Oriented and Relational Database Files," in Proceedings of the Fourth International Symposium on Large Spatial Databases, SSD'95, Springer-Verlag, August 1995.

Chakravarthy, S., et al. 1989, "HiPAC: A Research Project in Active, Time Constrained Database Management," Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, MA, July 1989.

Chakravarthy, S., Anwar, E., Maugis, L., and Mishra, D. 1994a, "Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules", Information and Software Technology, Vol. 36:9, September 1994, pp. 559-568.

Chakravarthy, S., Krishnaprasad, V., Anwar, E. and Kim, S. 1994b, "Composite Events for Active Databases: Semantics, Context, and Detection" in International Conference on Very Large Databases, VLDB, September 1994, pp. 606-617.

Chakravarthy, S. and Mishra, D. 1994c, "Snoop: An Expressive Event Specification Language For Active Databases", Data and Knowledge Engineering Journal, Vol. 14:10, October 1994, pp. 1-26.

Dayal, U., Buchmann, A., and McCarthy, D. 1988, "Rules are Objects Too: A Knowledge Model for an Active Object-Oriented Database Management System" in Proceedings of 2nd International Workshop on Object-Oriented Database Systems. Bad Muenster am Stein, Ebernburg, West Germany, September 1988.

DMA 1993, Military Standard: Vector Product Format. Draft Document No. MIL-STD-2407. Defense Mapping Agency, Fairfax, VA.

Dittrich, K., Kotz, A., and Mulle, J. 1986, "An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases" in Proceedings of ACM SIGMOD, September 1986.

Honnavalli, R. K. 1994, "Design and Implementation of an event based shop control application on Sentinel -- An Active Object-Oriented DBMS", MS thesis, Univ. of Florida, Industrial and Systems Engineering Department, August 1994.

Ishikawa 1990, "Object-Oriented Real-Time Language Design: Constructs for Timing Constraints" in OOPSLA '90 Proceedings, pp. 289-298.

Morgenstern, M. 1983, "Active Databases as a Paradigm for Enhanced Computing Environments" in Proceedings of International Conference on Very Large Data Bases, VLDB, pp. 34-42.

Shaw, K., et al. 1994, "Development of an Object-Oriented Digital MC&G Database System for Modeling and Simulation Support", Final Project Report to DMA and DMSO, Naval Research Laboratory, September 1994.

Su, S., Krishnamurthy, V., and Lam, H. 1988, "An object-oriented semantic association model" in Artificial Intelligence: Manufacturing Theory and Practice. The Institute of Industrial Engineers.

REPORT DOCUMENTATION PAGE

Form Approved
OBM No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1995	Proceedings

4. TITLE AND SUBTITLE

Implementation of a Rule-Based Framework for Managing Updates in an Object-Oriented VPF Database

5. FUNDING NUMBERS

Job Order No.

Program Element No. DMA

Project No.

Task No.

Accession No. DN16-3525

6. AUTHOR(S)

David K. Arctur*, E. Anwar*, S. Chakravarthy*, M. Cobb, M. Chung, K. Shaw, and J. Alexander*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Marine Geosciences Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/PP/7441--95-0052

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Mapping Agency
8613 Lee Hwy.
Fairfax, VA 22031-2138

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

GIS/LIS '95, November 14–16, 1995, Nashville Tennessee

*University of Florida

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper presents an approach for supporting reactive capability in an object-oriented GIS database, through the use of an event interface comprising an event generator and rule objects. This interface supports specification of events spanning sets of geographic-feature objects, and detection of primitive and complex events. Rules can be specified to apply either at a class level (i.e., to all instances of a given geographic-feature class) or at an instance level. In addition we allow evaluation of both pre- and post-conditions on changes to a feature. This approach is relevant in three distinct situations: (1) *immediate mode*, to execute rules immediately before or after some state change; (2) *deferred mode*, to execute rules at the end of several changes; and (3) *detached mode*, to perform rule-based actions separately from the state changes. GIS presents a rich set of problems for which this approach can be useful. This paper outlines the key elements of the rule-based approach employed in an object-oriented framework used for viewing and editing Vector Product Format (VPF) source data.

14. SUBJECT TERMS

GIS, Databases

15. NUMBER OF PAGES

11

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR